ADELE: Anomaly Detection from Event Log Empiricism

Subhendu Khatuya*, Niloy Ganguly*, Jayanta Basak†, Madhumita Bharde†, Bivas Mitra*
*Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, INDIA 721302
†NetApp Inc., Bangalore, India
Email: subhendu.cse@iitkgp.ac.in, niloy@cse.iitkgp.ernet.in, Jayanta.Basak@netapp.com,
madhumita.bharde@gmail.com, bivas@cse.iitkgp.ernet.in

***Abstract*—A large population of users gets affected by sudden slowdown or shutdown of an enterprise application. System administrators and analysts spend considerable amount of time dealing with functional and performance bugs. These problems are particularly hard to detect and diagnose in most computer systems, since there is a huge amount of system generated supportability data (counters, logs etc.) that need to be analyzed. Most often, there isn't a very clear or obvious root cause. Timely identification of significant change in application behavior is very important to prevent negative impact on the service. In this paper, we present ADELE, an empirical, data-driven methodology for early detection of anomalies in data storage systems. The key feature of our solution is diligent selection of features from system logs and development of effective machine learning techniques for anomaly prediction. ADELE learns from system's own history to establish the baseline of normal behavior and gives accurate indications of the time period when something is amiss for a system. Validation on more than 4800 actual support cases shows $\sim$ 83% true positive rate and $\sim$ 12% false positive rate in identifying periods when the machine is not performing normally. We also establish the existence of problem "signatures" which help map customer problems to already seen issues in the field. ADELE's capability to predict early paves way for online failure prediction for customer systems.**

***Index Terms*—System Log; Anomaly Detection**

I. Introduction

Reliable and fast support service in times of failure is a prerequisite for an efficient storage facility. Enterprise storage systems are typically used for mission-critical business applications. Customers consistently rely on storage vendors to provide high availability of data. Although failures cannot be completely avoided, a 24x7 support service that helps resolve issues within minimum case resolution time[1] is an imperative.

For efficient diagnosis of failures, complex enterprise systems have instrumentation to generate massive amounts of telemetry data everyday - typically in the form of counters, logs etc. When customer reports a problem (by opening a "support case") in the field, a support engineer with domain expertise tends to sift through weeks or even months of telemetry information to identify something abnormal. They typically rely on thumb rules (e.g. severity-based filtering for logs) or prior experience to identify relevant entries and thereby eventually, the root cause. However, our analysis (in Figure 1(a), distribution of resolution period (in days)) shows that about 50% of the cases take anytime between 3 to 20 days for resolution; mean resolution period is as high as 15 days whereas median values are 5-6 days. Evidently, this approach is not scalable, accurate or fast enough, especially when it comes to managing multiple node clusters in large data centers. Figure 1(b) demonstrates the fact that even when given a higher priority for triage, resolution periods do not improve, possibly due to complexity of such issues. Hence, improving the resolution period is one prime concern of the stakeholders.

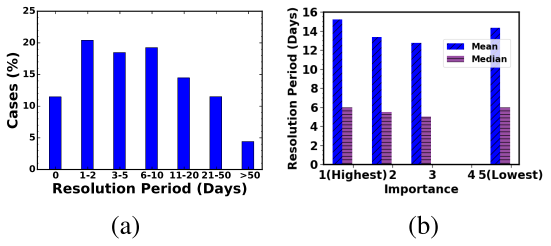


Fig. 1. Distribution of Resolution Period across (a) support cases (b) priority labels

The effectiveness of support service can be enhanced manifold if proactive detection of impending failures can be made automatically. However, the problem of proactive failure detection comes with multiple challenges. First of all, system failure is a complex process as it doesn't follow a single uniform pattern. Since the problems observed due to a *system bug* is an artifact of the combination of anomalous behaviors of the multiple subsystems, the resultant failure patterns exhibit diversity across different bugs. Moreover, the contribution of each subsystem (e.g raid, wafl etc.) towards failure depends on the properties of that specific bug. Most of the state of the art endeavors have overlooked the problem of bug specific anomalous behaviors exhibited by different subsystems. Secondly, these anomalies have several categories and the problems slowly manifest in variety of ways. For example, system misconfigurations are known to develop failures over time. File-system fragmentation [13] and misaligned operations cause performance slowdown. Such logged events are of varying severity (e.g. warning, info, etc.) and often go unnoticed or are ignored. However, these obscure malfunctions

[1]Case resolution period is the duration between 'Case opened date' and 'Case closed date' from customer support database.

from one subsystem propagate and cascade to others and result in an undesirable behavior from an overall system or application perspective. Thirdly, their is a subtle difference between anomaly and failures observed. During operation, subsystems or modules might exhibit anomaly such as system slowdown due to heavy load, increase response time of storage I/O, however that may not always necessarily lead to failure. Proper handling of those (false) signals is important as this may raise a large numbers of false positive alerts which may slower the utility of the failure prediction system. This paper takes an important step towards this direction.

In terms of data sources for anomaly identification, counters and logs provide different value propositions. Counters are collected periodically while logs have event driven characteristics. In general, system performance issues can be better diagnosed with the help of counters [11] [10]. On the other hand, for issues like system misconfiguration or component failures, logs contain valuable signals for prediction of anomalies [12][8][2]. There have been several studies on anomaly detection from the log files; for instance Liang et al [9] proposed the methodology to forecast the *fatal* event from IBM Bluegene/L logs. Jiang et al [5] provided interesting data-driven analysis of customer problem troubleshooting for around 100000 field systems. Shatnawi et al [14] proposed a framework for predicting online service failure based on production logs.
This paper effectively highlights the challenges of the noisy log events and conducts an extensive log analysis to uncover the anomaly signatures. We start with background about customer support infrastructure and data selection criteria (Section II). Subsequently, we develop ADELE, which leverages on the machine learning techniques to (a) predict system failure in advance by analyzing of log data and anomaly signatures generated by different modules (Section IV) and (b) develop an automated early warning mechanism to protect the system from catastrophe. In case of a massive failure, the final failure is typically preceded by a sequence of small malfunctions, which most of the time have been overlooked. However, if correctly diagnosed at proper time, these apparently harmless malfunctioning signals can predict, in advance, the occurrence of a big failure. The novelty of ADELE is manifold (a) ADELE captures the uniqueness of the individual bugs; while detecting the anomaly, it considers and estimates the (varying) responsibility of the individual subsystems causing the problem (b) instead of merely relying on the vanilla features for anomaly detection, ADELE observes the abnormality exhibited by the individual features (via outlier detection) to compute the anomaly score (c) finally, through empirical experiments, we discriminate the failures with the anomaly, which substantially reduces the false alarms.
We show that ADELE outperforms the baseline with 83% True positive and 12% False Positive rate. It should be noted that the log analysis method presented here is a black-box method and does not tie itself to any domain-specific context.

In a nutshell, ADELE makes following major contributions:

- A comprehensive and generic abstraction of storage system logs based on their metadata characteristics is developed (Section III).
- Through detailed study and rigorous analysis, the relationship between anomaly signals and failure is established. Problem-specific models are learnt using machine learning techniques and accuracy is demonstrated with large number of customer support cases (Section V).
- Problem specific models and signatures are extended for generic failure prediction by mapping unknown problems to already seen issues in field. More importantly, ADELE creates groundwork for an proactive, online failure prediction (Section VI).

Field	**Log Entry Example**	**Description**
Event Time	Sat Aug 17 09:11:12 PDT	Day, date, timestamp and timezone
System name	cc-nas1	Name of the node in cluster that generated the event
Event ID	filesystem1.scan.start	EMS event ID. Contains Subsystem name and event type
Severity	info	{Severity of the event}
Message String	Starting block reallocation on aggregate aggr0	Message string with argument values

TABLE I
EMS MESSAGE STRUCTURE

II. BACKGROUND AND DATASET

A. Auto Support

System generated data like counters, logs and command history etc. are critical for troubleshooting of customer issues. Auto Support (ASUP) infrastructure provides an ability to forward this logged data (storage system log) daily to NetApp. While customers can opt out of this facility, most of them choose to go for it since it provides proactive system monitoring capabilities and faster customer support.

B. Event Message System (EMS) Logs

Support infrastructure described above gives access to daily EMS logs[2]. An example of a typical EMS log entry is as follows:
Sat Aug 17 09:11:12 PDT [cc-nas1:filesystem1.scan.start:info]: Starting block reallocation on aggregate aggr0.
Interpretation of fields is summarized in Table I in context of the above log entry. Each log entry contains time of the event with fields like day, date, timestamp and timezone. As multiple systems (alternatively, storage "nodes") are clustered in a typical Data ONTAP® setup, name of the node where event occurred within the cluster is required to be part of the log entry. Event ID gives information about the kind of event that occurred in an hierarchical fashion. First part of this ID is the subsystem that generated the event. Severity field can take a value from 'emergency', 'alert', 'critical', 'error', 'warning', 'notice', 'info', 'debug'. Message string throws more light on the event by incorporating context-specific argument values (e.g. 'aggr0' from above example) for describing the event.

[2]https://library.netapp.com/ecmdocs/ECMP1196817/html/event/log/show.html

C. *Customer Support Database and Bug Database*

Customer Support Database: Customer support portal provides customers an ability to report their grievances. It also provides a forum for customers to engage with support engineers to clarify configuration queries and provide other guidance. Cases can be system generated based on predefined rules or human-generated. We query this database for fields like opened date, closed date, priority label etc. for customer support cases.

Bug database: Bug database is typically internally oriented and tracks the engineering side of problems. The same bug can be experienced across multiple customers, systems and configurations. We use this database to query customer cases associated with bugs.

D. *Data Filtering*

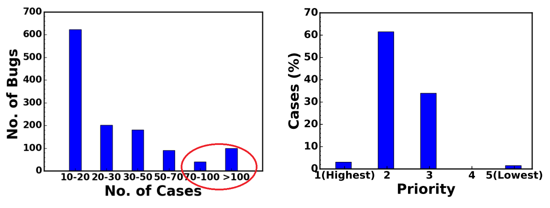
For evaluation of the methodology, we selected most severe and impactful cases filed by customers. Customer cases can get escalated and are filed as bugs with bug database. Figure 2(a) shows distribution of number of cases across bugs. We filter dataset of cases and bugs in four steps as following:

(a) First, we select bugs having a sufficient number of cases associated with them. This ensures sufficient data to validate the model and also provides robustness and generality to the mechanism as these cases are spread across multiple systems, customers and configurations.

(b) In second step, we filter out instantaneous failures - bugs that are race conditions[3], coding errors etc. These are failures that cannot be predicted and are not in "slow death" category.

(c) As a third step, we filter cases based on the priority label. These are the most severe and impactful cases filed by customers.

(d) In the last step, we filter out cases with missing data. In a five month time period required for validation, we exclude those cases from analysis if system logs aren't available for one or more days.



(a) (b)

Fig. 2. (a) Case distribution across bugs (b) Priority distribution across cases

As per the above criteria, we identified 48 bugs with at least 70 customer support cases associated with each of them (Figure 2(a)). We concentrated on cases with priority labels 1, 2, 3 (Figure 2(b)). Finally, we analyzed 4827 cases for about 4305 unique systems that span over 5.5 (January 2011 to June 2016) years.

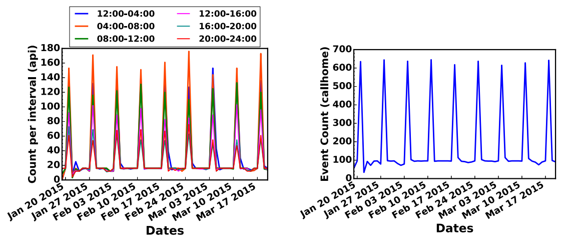
[3]https://en.wikipedia.org/wiki/Race_condition

III. Anomaly detection: Attributes and its representation

In this section, we first demonstrate the evidences showing the potential of event logs, generated by multiple subsystems, to detect overall system anomaly. Next, we perform rigorous experiments uncovering the features exhibiting the anomaly signatures. Finally we propose a novel representation of the features with the help of score matrix.

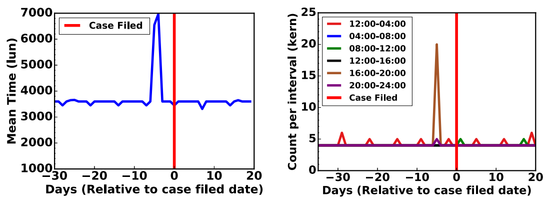
A. *Periodicity and Anomaly Clues*

Attributes of event logs generated by a subsystem seem to show a weekly periodic pattern. Some typical examples are shown in Figures 3 & 4. In Figure 3(a), number of events generated by "api" subsystem shows an almost perfect periodic pattern across different time intervals of the day. Similarly, amount of total messages by "callhome" module show a weekly recurring pattern. The recurrence is expected, typically due to planned maintenance, scheduled backups, workload intensity changes [7] - e.g. exchange server has similar activity profile across same days of the week - Mondays show a distinctly different behavior from Sundays.



(a) (b)

Fig. 3. Periodicity (a) Event count across intervals for "api" (b) Event count for "callhome"



(a) (b)

Fig. 4. Anomaly (a) Mean inter-arrival time for "lun" (b) Event count across intervals for "kern"

If one or more subsystems are going through an anomalous phase, it gets reflected in some attributes of logs generated for those subsystems. Red vertical lines on Figures 4(a) & 4(b) represent the date when customer reported an issue. We can see that mean inter-arrival time between messages for "lun" module (Figure 4(a)) shows an irregular behavior before a customer reports the problem. Since there is no practical and standard way of knowing the exact time a failure occurred, we used reported time as the metric- since it can be safely assumed that that's the time when failure was first sensed. Also, number of events generated by "kern" module (Figure

4(b)) across different intervals in the day shows an abnormal behavior few days prior to case filed date. A system failure is typically preceded by one or more modules generating events in patterns that are not "normal"[4].

B. Engineering Log Attributes

EMS logs have a total of approximately 7800 event types spanning 331 subsystems. We extract 18 attributes corresponding to each subsystem as summarized in Table II. Figure 5 shows a representative example of each attribute showing an anomaly signal before case is filed by the customer. As the range of values are different for different subsystems, we normalized the values between 0 and 1.

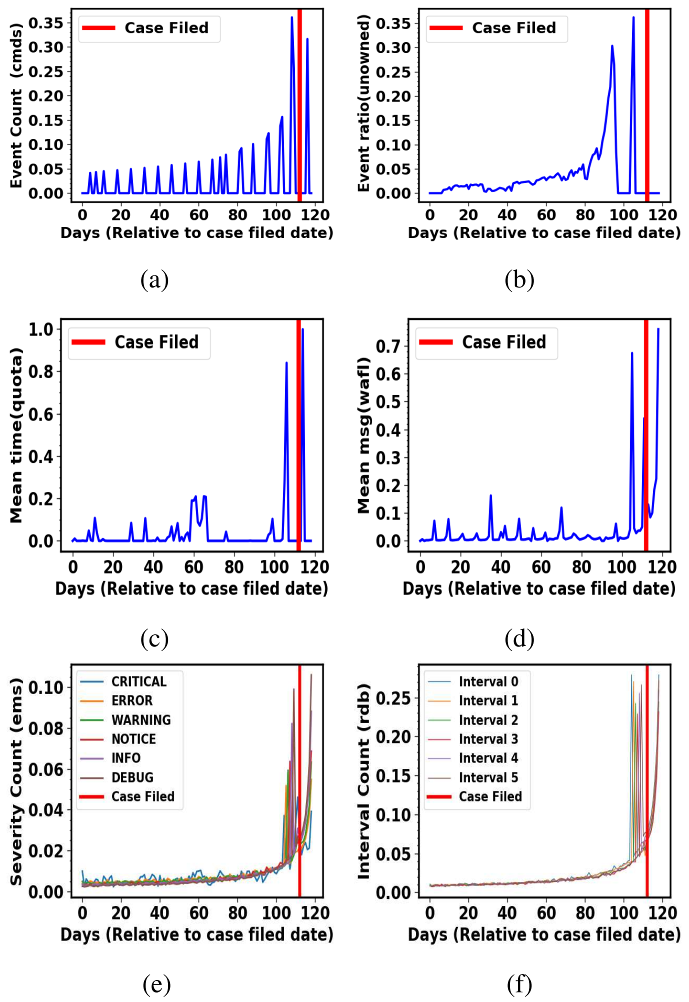


Fig. 5. (a) Event count of *'cmds' subsystem (b) Event ratio of* 'unowned' subsystem (c) Mean time of *'quota' module (d) Mean msg of* 'wafl' module (e) Severity Count of *'ems' subsystem (f) Interval Count of* 'rdb' module.

1) **Event Count:** Event count denotes the number of events generated by the particular subsystem. For instance the event count of 'cmds' module (Figure 5(a)) shows an irregular behavior before case filed date.

[4]We have used the term *anomaly* to signify deviant behavior for a day while *failure* to signify it over an extended period (couple of days/one week) - the rationale being that failure sets in when anomaly persists for some time.

2) **Event Ratio:** Event Ratio is the fraction of events generated by the particular subsystem. The irregular behavior of 'unowned' module is observed in Figure 5(b).
3) **Mean inter-arrival time:** We define Mean inter-arrival time as the mean time between successive events for that subsystem. Figure 5(c) is a representative instance of subsystem 'quota' showing an irregular behavior before case filed.
4) **Mean inter-arrival distance:** The Mean inter-arrival distance is defined as mean number of events by other subsystems between successive events of module under consideration. Figure 5(d) shows an example where we have shown the behavior of 'wafl' which seems anomalous before case filing.
5) **Severity Spread:** We capture severity spread for the subsystem in 8 attributes (Severity 0 to 7) with severity values 'EMERGENCY (Severity 0)', 'ALERT', 'CRITICAL', 'ERROR', 'WARNING', 'NOTICE', 'INFO', 'DEBUG (Severity 7)' each. The severity spread of 'ems' subsystem is shown in Figure 5(e).
6) **Time-interval Spread:** To capture activity profile of different time intervals during the day, we define six intervals that represent event counts during the interval for the particular subsystem. As an example we have shown only the activity profile at each interval of 'rdb' subsystem in Figure 5(f).

Attributes like severity spread and time-interval spread detect changes in individual module's behavior. On the other hand, event ratio, mean inter-arrival time and mean inter-arrival distance capture normalcy of a module relative to other modules. We need a holistic approach in detecting system failure with certainty. Our hunch is that false positive rate will be really high if we treat every irregular signal for every subsystem ("anomaly") as a criterion for predicting failure. It Would be better if we could prove that with some data. It should be noted that attributes listed in Table II can be extracted from most type of logs, and are not specific to EMS.

Attribute	**Description**
Event Count	Total number of events
Event Ratio	Ratio of number of events to total number of messages
Mean Inter-arrival Time	Mean time between successive events
Mean Inter-arrival Distance	Mean number of other messages between successive events
Severity Spread	Eight attributes corresponding to event counts of each severity type
Time-interval Spread	Six attributes denoting event counts during six four-hour intervals of the day

TABLE II
ATTRIBUTES EXTRACTED PER SUBSYSTEM (MODULE)

Table III shows how anomaly signals are distributed across attributes for our dataset. Interestingly and intuitively, except lower severity (0,1,2) values, most attributes have sufficient clues for predicting system anomaly. Importantly, average number of days relative to case filed date when these anomaly signals appear is ranging from 6-12. Both these are highly encouraging starting points for leveraging these log attributes for overall anomaly detection for the system. In another high-level empirical study described in Table IV, we observe the distribution of anomaly signals across susbsystems(module). This Table shows statistics for only top 7 subsystems across

Attribute	Cases (%)	Total Signals	Average Days
Event Count	68.39	7163	10.86
Event Ratio	66.25	7037	11.01
Mean Inter-arrival Time	80.30	12955	9.90
Mean Inter-arrival Distance	66.10	9816	11.08
Severity 0	0.00	0.00	0.00
Severity 1	0.30	14	10.0
Severity 2	0.00	0.00	0.00
Severity 3	8.39	456	11.88
Severity 4	14.65	773	10.91
Severity 5	12.06	648	11.23
Severity 6	34.04	2203	10.91
Severity 7	42.59	3124	12.89
Interval 1	46.87	4016	7.25
Interval 2	44.58	3249	9.95
Interval 3	48.24	4045	10.76
Interval 4	47.63	3854	6.80
Interval 5	52.06	4178	11.67
Interval 6	49.00	4289	12.13

TABLE III
DISTRIBUTION OF ANOMALY CLUES ACROSS ALL CASES. NOTE THAT EXCEPT LOWER SEVERITY VALUES (0,1,2), MOST ATTRIBUTES SHOW SUFFICIENT CLUES.

all log attributes described earlier. Reasonably high number of cases show subsystems in anomalous conditions considerably early from the case filed date.

Subsystem	Cases (%)	Total Signals	Average Days
raid	13.03	8842	10.82
kern	11.51	7811	10.98
wafl	7.95	5394	11.01
ems	7.12	4834	11.10
callhome	7.09	4811	7.26
disk	5.21	3537	10.87
hamsg	3.94	2674	10.92

TABLE IV
DISTRIBUTION OF ANOMALY SIGNALS ACROSS SUBSYSTEMS.

C. Representation of Features: Log Transformation

Extracted 18 attributes corresponding to each subsystem (module) per day is summarized in Table II. We call each Module-attribute combination as "feature". First, we represent EMS log of d^{th} day as a raw feature matrix as follows.

$$X_d = \left\{ \; X_{i,j}^{(d)} \text{ where } i \in M \text{ and } j \in A \; \right\}$$

where M is the set of modules ($|M| = m$) and A is the set of attributes ($|A| = a$). i^{th} row and j^{th} column of X_d contains j^{th} attribute's value of i^{th} subsystem.

IV. ADELE: Framework For Anomaly Detection

The proposed methodology computes an overall system anomaly score for a candidate case. Figure 6 presents an overview of the core methodology of ADELE. We group the support cases based on their association with bugs and we train the model for each bug. We collect EMS logs for 5 months for each support case (18 weeks before and 2 weeks after case filed date). Log of each day is then transformed into the corresponding score matrix, as a representation of the features. We present the machine learning technique to estimate system anomaly score from the computed features. The coefficient vector considered as problem signature is the model learnt.

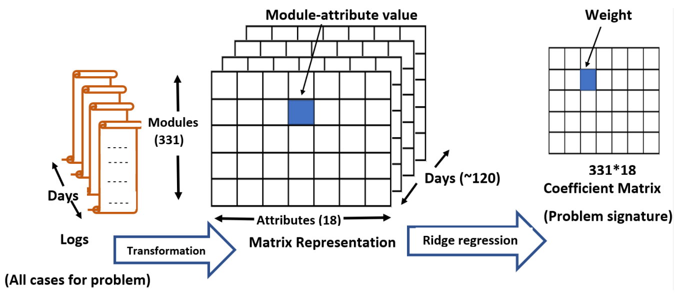


Fig. 6. Overview of the Anomaly identification Process

A. Anomaly Scores For Individual features

We transform the raw matrix (X_d) to an equivalent score matrix which captures the abnormality observed by the individual features. In section III-A, observed periodicity of log attributes corresponds to normal behavior whereas deviations from periodic behavior provide clues to anomalous behavior. With these observations, we fit normal distribution with the moving window of feature values over last few weeks - e.g. ($X_{i,j}^{(d-7)}$,$X_{i,j}^{(d-14)}$,$X_{i,j}^{(d-21)}$, ...). Let $\mu_{i,j}$ be the mean and $\sigma_{i,j}$ be the standard deviation for a particular observation $X_{i,j}$ across last four weeks. CDF (Cumulative Distribution Function) of normal distribution [1] is given by

$$CDF(X_{i,j}) = \frac{1}{\sigma_{i,j}\sqrt{2\pi}} e^{-(X_{i,j}-\mu_{i,j})^2/2\sigma_{i,j}^2} \tag{1}$$

Anomaly score of this observation $X_{i,j}$ is then calculated as

$$S_{i,j} = 2 * |0.5 - CDF(X_{i,j})| \tag{2}$$

which is a measure of outlier [4]. This captures how far the value deviates from normal. Thus, we transform the matrix (X_d) of d^{th} day into a score matrix (S_d) using above formula for each feature.

B. System State Labeling

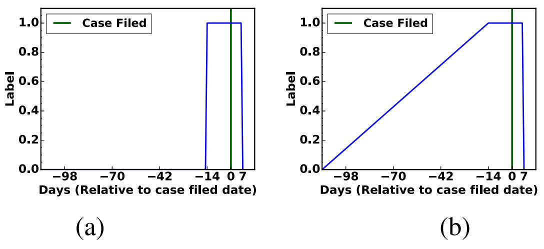


Fig. 7. Labelling (a) Step (b) Ramp.

In order to label the state of the system for training and testing, we consider observations in section III-A. From our analysis, anomaly signals typically start appearing about 2 weeks before case is filed (Refer Table III, IV). Also, the system can be considered to be in abnormal state till the case is resolved (refer Figure 1, median case resolution time is 5-6 days.) With these data-driven insights, we label 2 weeks before and 1 week after case filed date as an abnormal period. Here data produced in each of those days are marked as anomalous. Rest of the days are treated as normal. In Figure 7, we consider two types of labeling strategy (a) Step and (b) Ramp. In step labeling, we assign a score of either 0 (normal) or 1(abnormal). In case of ramp labeling, system state of each day is annotated with a fractional number, the number is inversely proportional to the number of days before failure set in - that is further the day is from failure smaller is the value assigned.

C. Estimating System Anomaly Score

We develop ADELE to compute the overall anomaly score from the feature vector represented as the score matrix S_d. Each feature ($S_{i,j}$) contributes differently to overall anomaly of the system depending upon the specific problem. For instance, disk subsystem might be highly anomalous for some problems but not for others. In essence, these contributions are problem-specific and are learnt using Ridge regression. Two labelling strategies as described in section IV-B are used for learning. We perform learning using Ridge regression[5] which minimizes squared error while regularizing the norm of the weights. It performs better as compared to other methods because of its inherent mechanisms to address the possibility of multi-collinearity and sparseness of coefficients which is typical in our dataset. The weight vector ($\mathbf{w}$) of length $m * a$ returned by Ridge regression is termed as *coefficient vector*. Here we consider score matrix S_d of dimension $M \times A$ as feature matrix. The loss function ($J(\mathbf{w})$) of Ridge regression can be represented as follows.

$$J(\mathbf{w}) = \lambda \left|\left|\mathbf{w}^2\right|\right| + \sum_i (w^T \cdot \mathbf{s_i} - y_i)^2. \tag{3}$$

where $\mathbf{s_i}$ is input vector, y_i represents the corresponding output label of observation i and λ is the shrinkage parameter. Then the stationary condition is

$$\frac{\partial J}{\partial \mathbf{w}} = 0 \tag{4}$$

$$(S_d^T + \lambda I)\mathbf{w} = S_d \mathbf{y} \tag{5}$$

$$\mathbf{w} = (S_d S_d^T + \lambda I)^{-1} S_d \mathbf{y} \tag{6}$$

Finally after training the model with feature matrix (S_d) and ground truth label ($\mathbf{y}$) the Ridge regression estimates the system anomaly score.

Anomaly Signature: Overall weight vector ($\mathbf{w}$) learnt through Ridge regression, denotes the relative importance of different features in identifying the problem (bug) correctly. This vector becomes the *signature* of the problem.

V. Evaluation

In this section, we perform a 5-fold cross-validation over all cases associated with each of the 48 bugs and compute the corresponding coefficient vector $\mathbf{w}$. We apply Ridge regression with regularization parameter $\alpha = 0.5$ and tolerance$= 0.01$. We select the value of α and tolerance empirically which essentially reduces the variance of estimates and increases the precision of the solution.

[5]https://onlinecourses.science.psu.edu/stat857/node/155

A. First glimpse

Figure 8 illustrates the aggregated anomaly scores as computed by our formulation for 4 different cases belonging to four different bugs. Note that these scores cross threshold value thereby giving an early signal of anomaly which ultimately will lead to failure.

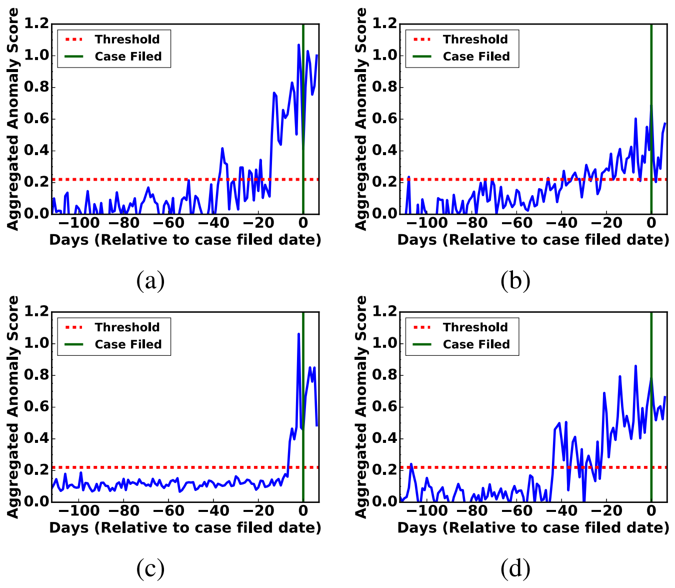


Fig. 8. ADELE's aggregated anomaly scores exceed threshold several days before case is filed

B. Performance Metrics

We define the standard yardsticks in this context. **True Positives (TP):** Anomaly that are correctly identified. **False Positives (FP):** Normal days that are flagged as anomalous. **True Negatives (TN):** Normal days that are flagged as normal. **False Negatives (FN):** Anomalous days that are not identified correctly. We calculate: **True Positive Rate (Recall)** $TPR(\%) = \frac{TP}{TP+FN} \times 100$. **True Negative Rate (Specificity)** $TNR(\%) = \frac{TN}{TN+FP} \times 100$. **False Positive Rate (Fall-out)** $FPR(\%) = 100 - TNR(\%)$.

C. Model Performance

In a moving window of *base period* (last 7 days), if anomaly score exceeds threshold for a certain number of days (3 days), we flag the onset of failure. (These choices are explained later in section V-F). Figures 9(a),(b) show TPR(%) plotted on X-axis and FPR plotted on Y-axis for step and ramp labeling respectively. In this scatter plot, each point represents a bug plotted considering corresponding average TPR(%) and FPR(%) values calculated across all of its support cases. Most bugs are concentrated near top left corner, as expected ideally. From Figure 9, it is seen that step labelling performs much

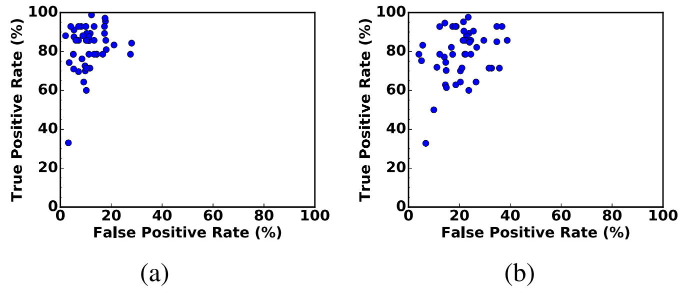


Fig. 9. TPR-FPR plot with (a) Step (b) Ramp Labelling; Step labelling performs better than Ramp labelling.

better as compared to ramp labeling as the points in the latter are less concentrated signifying inferior TP and higher FP.

D. Early Detection

Due to high fidelity of the system, the failure can be captured as soon as it sets it (14 days prior to case file date as per our design). Whatever is missed is captured in the subsequent days; Figure 10(a) exhibits the distribution of those cases. The study shows ADELE can in most of the cases set in warning much before actual failure sets in.

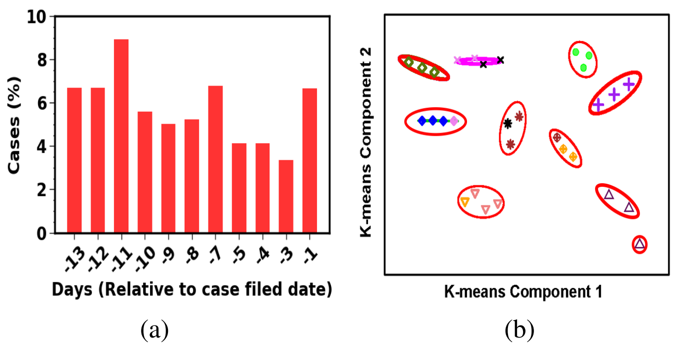


Fig. 10. (a) Early detection of failure by ADELE model (b) The problem signatures are consistent across subgroups of cases for same bug

E. Anomaly Signatures

An important sanity check of our system would be to inspect whether all the cases belonging to a specific bug exhibit similar anomaly signatures under the proposed framework. To demonstrate this, we consider 10 bugs and split cases for each bug in 3 non-overlapping subgroups. Ridge regression is applied on each of these subgroups separately to get coefficient vector ($\mathbf{w}$) which become corresponding anomaly signatures. (Dis)-similarity between all pairs of signatures is calculated using Euclidean distance. We then use K-means [3] algorithm ("clusplot"[6] method in R) to cluster these signatures. Figure 10(b) shows that different subgroups of cases for same bug (represented by same color) get clustered together (represented by rings) because of similarity. When we extend this to all bugs (48) studied, only 23 (out of 48 $\times$ 3) subgroups are misclassified. This indicates that signatures are similar for the same bug irrespective of the cases used to learn the model and largely distinct from other types of bugs.

[6]https://stat.ethz.ch/R-manual/R-patched /library/cluster/html/clusplot.default.html

F. Selection of Parameters

True positives and false positives typically exhibit a classic trade-off as they show similar behavior (both TP and FP either reduce or increase) when tunable parameters are varied. As part of the design, we would like to achieve high TP while allowing as minimum FP as possible. A high FP would falsely ring the alarm bell several number of times, thus eroding the confidence of the maintenance engineer using the system. To quantitatively determine the ideal point, we plot TPR and TNR (100-FPR) for varying value of the parameter under consideration - the point of intersection between these curves is considered the optimal point from design perspective.

We obtain the optimal values of the following three parameters pursuing the protocol.

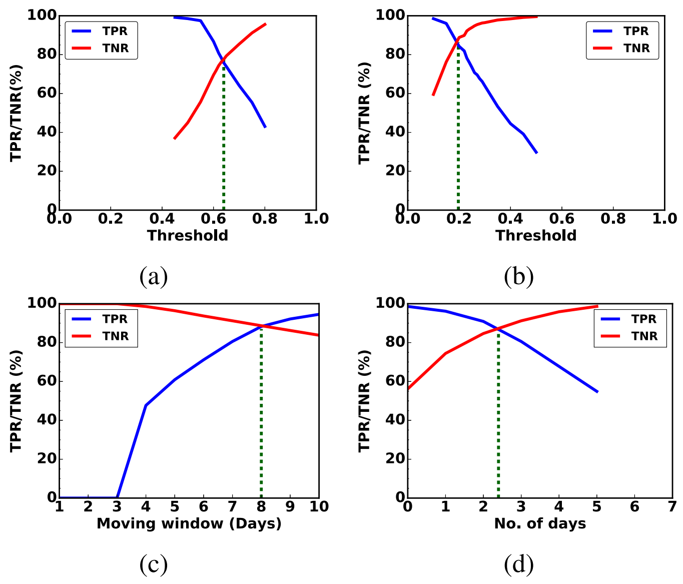


Fig. 11. Threshold Selection: (a) 0.70 and (b) 0.22 are optimal values for ramp and step labeling respectively. (c) Base period: 7 days (d) Critical period :3 days are optimal values.

(a) **Threshold:** The optimal threshold is approximately 0.7 and 0.2 for ramp labelling and step labelling respectively (Figure 11(a), (b)).
(b) **Base Period:** We choose a moving window of base period for assessment. From domain knowledge of weekly patterns (section III-A), 7 days seemed like an ideal base period to assess the anomaly. From TPR-TNR plot in Figure 11(c), 8 days (close to expected value of 7) is the optimal point.
(c) **Critical Period:** This is the number of days in base period for which anomaly scores exceed threshold value. Selection of this critical period as 3 is justified in TPR-TNR plot in Figure 11(d).

Note that, since these three parameters have dependency among themselves, the initial values are guessed checking a randomly chosen subset of cases and then the values are fixed by running the experiments iteratively.

G. Comparison with Baseline Model

We implement the following three baseline models to compare the performance of our model. The first two baselines are borrowed from the state of the art competing algorithms whereas the third one is the variation of ADELE.

(a) Liang Model: Prediction methodology of [9] involves first partitioning the time into fixed intervals, and then trying to forecast whether there will be failure events in each interval based on the event characteristics of preceding intervals. This method studies patterns of *non-fatal* and *fatal* events to predict *fatal* event in BlueGene systems. Following two assumptions have been made (i) We treat case filing action as *fatal* event. (ii) As per terminologies from this paper, we choose observation window of 7 days and 7th and 8th day as the current and prediction window, respectively.

As prescribed in their work, a total of 8265 features are extracted from EMS log. Using step labeling, we apply recommended procedure - SVM with RBF kernel over this feature-set for prediction of *fatal* event (customer reporting a problem, in our case).

(b) EGADS Model: Laptev et. al. [6] introduces a generic and scalable open source framework called EGADS (Extensible Generic Anomaly Detection System) for automated anomaly detection on large scale time-series data. EGADS framework consists of two main components: the time-series modeling module (TMM) and the anomaly detection module (ADM). We consider three specific implementations of ADM namely KSigma, DBScan and ExtremeLowDensity. For the sake of comparison, (i) The abstracted raw matrix (Section III-C) of each day (treated as timestamp) is converted into a multivariate row vector to build the input as time-series data. (ii) Due to multivariate nature of our input data we have chosen multiple regression model as TMM.

(c) ADELE Direct: In this baseline model, we have implemented a variation of the ADELE where we directly use raw matrix features. Considering the same labeling (Section IV-B), we apply Ridge regression and perform 5-fold cross-validation.

Model	TPR (%)	FPR (%)	Accuracy (%)	F1 Score (%)
Liang	74	34.75	66.79	0.440
EGADS-KSigma	***76.41***	22.49	***77.31***	***0.543***
EGADS-DBScan	59.52	***20.05***	76.34	0.476
EGADS-ExtremeLowDensity	62.31	21.78	75.41	0.475
ADELE Direct	47.87	27.04	68.53	0.352
ADELE	**83.92**	**12.02**	**87.26**	**0.710**

TABLE V
COMPARISON WITH BASELINE MODELS. ADELE BEATS ALL THE BASELINE IN TERMS OF ALL PERFORMANCE METRICS.

Evaluation of ADELE against baselines: We calculate the TPR and FPR for the aforesaid baseline algorithms, as described in Section V-B. From the Table V we observe that ADELE beats all the baseline handsomely. Figure 12(a) shows the early detection result of ADELE and EGADS-KSigma model. Our model detects failure early in more cases as compared to EGADS model. From Figure 12(b), we observe that the bug specific TPR-FPR result of ADELE is also better than EGADS. The result of different anomaly detection modules (ADM) proposed in [6] is shown in Table V. Only EGADS K-SigmaModel technique produces best result (76.41% TPR and 22.49% FPR overall) among all other ADM, whereas [9] beats the remaining ADM of EGADS. Performance exhibited by ADELE Direct model is the poorest since it takes a vanilla approach of feeding raw features into a ML model; whereas ADELE uses a novel score matrix formulation enabling it to achieve superior performance.

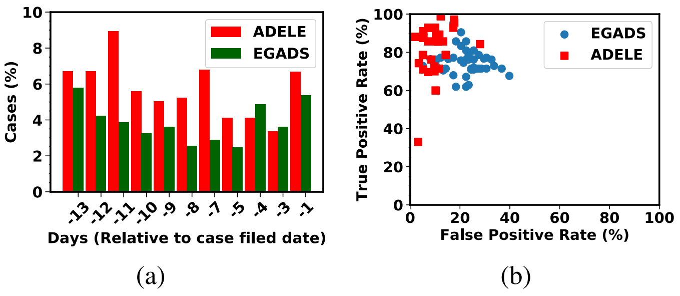


Fig. 12. (a) In more cases ADELE detects failure early as compared to EGADS. (b) TPR-FPR plot for each bug using EGADS-KSigma model and ADELE model

VI. ADELE:ONLINE FAILURE PREDICTION

Finally we extend the ADELE to develop a framework for predicting failures in real time. The focus of this online failure prediction is to perform short-term failure predictions based on current state of the system. Here we trace the system log at each specific time interval (1 day taken in our model) and notify the health of the system; the system needs to be trained for at least 1 month to construct the score matrix. The schematic diagram of our proposed real time failure prediction model is shown in Figure 13. Given an (unknown) system log, the major challenge is to identify the correct weight vector which carries the proper signature of the bug. Since in our dataset, we explore the most frequently occurring 48 bugs, we expect to map the unknown log to one known bug. Additionally, in practice, this narrows down the probable candidates for the support person and with the help of domain knowledge, he can quickly nail down the correct issue. The outline of the approach is described below.

A. Mapping to A Known Bug

We build a dictionary (key-value pair) taking bug ID as the key and coefficient vector ($\mathbf{w}$) as the value. Our goal is to map a random case (bug is unknown) to any of the known bugs (amongst 48). For the customer case to be mapped to a known issue, we follow the same procedure described earlier to transform daily logs to a score matrices (S_d) and estimate system anomaly scores for each anomaly signature present in the dictionary. The estimated anomaly score is the sum of linear combination of score matrix and coefficient vector (inner

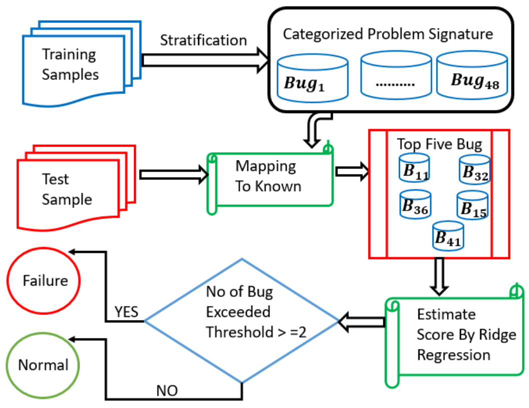
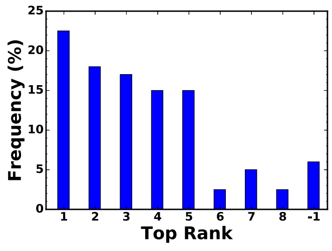
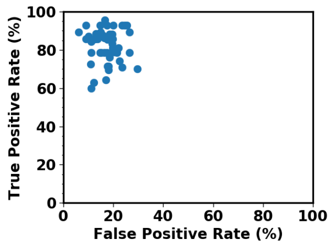


Fig. 13. A schematic of our proposed framework for online failure prediction.

dot product) and intercept values. We separately calculate intercept values (difference of actual Ridge function estimation and inner dot product result) for each bug. Finally, we rank all the bugs based on the estimated anomaly score and select top 3-5 ranked bugs. In actual practice, this essentially makes the troubleshooting process semi-automated since it narrows down the probable candidates for the support personnels. They can quickly nail down the correct issue further with domain knowledge. For validation of our approach, we expect the correct bug to appear in top five ranks. Figure 14(a) shows that around 87% cases ground truth matches with one of the top five ranks given by ADELE. In only 6% cases, the actual bug is ranked beyond 8.



(a)



(b)

Fig. 14. (a) ADELE correctly predicts the bug in top 5 for 87 % cases (b) Case specific TPR-FPR result by online failure prediction model

B. Predicting System Failure

Finally, we estimate the anomaly score of each day taking the top five coefficient vectors (corresponds to the top five mapped bugs) and score matrix of the corresponding day into consideration. We count the number of cases (out of top 5), whose estimated score exceeds the threshold; exceeding the threshold for the 50% cases predicts the failure in the system. For validation, we used over 49 customer reported cases. Result for a particular case over (multiple) days is shown in Figure 14(b). Overall we get **82.26%** as average TPR and **17.10 %** as average FPR.

VII. Conclusion And Future Work

In this paper, we have provided insights into storage system logs via extracted attributes. We used machine learning techniques to identify anomalous signatures from this construction for already known problems. In this process, we could reasonably accurately identify anomaly and beat baseline handsomely. After building the model for known bugs, we proposed a mixed model which is capable to map unknown cases to known bug and detecting failure in real time. ADELE can identify abnormality in the system roughly 12 days early. This essentially means that if support person can intervene in any one of those 12 days, failure can be avoided. We observe that *signatures* are consistent; hence similar type of problems can be identified through the proximity of their underlying signatures and a automated method of root cause analysis can be developed - this would be our future work.

ACKNOWLEDGEMENT

This work has been partially supported by NetApp Inc, Bangalore funded research project “Real Time Fault Prediction System (RFS)”. The authors also extend their sincere thanks to Ajay Bakhshi and Siddhartha Nandi from Advanced Technology Group, NetApp for facilitating this research.

References

[1] W. J. Dixon, F. J. Massey, et al. *Introduction to statistical analysis*, volume 344. McGraw-Hill New York, 1969.
[2] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White. A meta-learning failure predictor for blue gene/l systems. In *ICPP 2007*, pages 40–40. IEEE, 2007.
[3] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *J. Royal Stat. Soc. Series C (Applied Statistics)*, 28(1):100–108, 1979.
[4] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.
[5] W. Jiang, C. Hu, S. Pasupathy, A. Kanevsky, Z. Li, and Y. Zhou. Understanding customer problem troubleshooting from storage system logs. In *FAST*, volume 9, pages 43–56, 2009.
[6] N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD*, pages 1939–1947. ACM, 2015.
[7] N. Li and S.-Z. Yu. Periodic hidden markov model-based workload clustering and characterization. In *CIT 2008. 8th IEEE International Conference on*, pages 378–383. IEEE, 2008.
[8] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta. Filtering failure logs for a bluegene/l prototype. In *DSN 2005)*, pages 476–485. IEEE, 2005.
[9] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. Failure prediction in ibm bluegene/l event logs. In *ICDM 2007*, pages 583–588. IEEE, 2007.
[10] V. Mathur, C. George, and J. Basak. Anode: Empirical detection of performance problems in storage systems using time-series analysis of periodic measurements. In *MSST 2014*, pages 1–12. IEEE, 2014.
[11] V. Nair, A. Raul, S. Khanduja, V. Bahirwani, Q. Shao, S. Sellamanickam, S. Keerthi, S. Herbert, and S. Dhulipalla. Learning a hierarchical monitoring system for detecting and diagnosing service issues. In *Proceedings of the 21th ACM SIGKDD*, pages 2029–2038. ACM, 2015.
[12] F. Salfner and S. Tschirpke. Error log processing for accurate failure prediction. In *WASL*, 2008.
[13] M. Seltzer, K. A. Smith, H. Balakrishnan, J. Chang, S. McMains, and V. Padmanabhan. File system logging versus clustering: A performance comparison. pages 21–21. USENIX Association, 1995.
[14] M. Shatnawi and M. Hefeeda. Real-time failure prediction in online services. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1391–1399. IEEE, 2015.